IRIDIS ALPHA THEORY

ROB HOGAN

Contents

A Hundred Thousand Billion Theme Tunes

The theme music in Iridis Alpha is procedurally generated. There isn't a chunk of music data that the game plays every time you visit the title screen. Instead a new tune is generated for every visit. There's a distinction to be made here between procedural and random. The music isn't random: the first time you launch Iridis Alpha, and every subsequent time you launch it, you will hear the same piece of music. But as you let the game's attract mode cycle through and return to the title screen you will hear a new, different piece of music. Iridis Alpha has an infinite number of these tunes and it plays them in the same order every time you launch it and as it loops through the title sequence waiting for you to play.

Because the music is generated procedurally, and not randomly, you will hear the same sequence of tunes every time you launch the game so it appears to you as if the music was composed in advance and stored in the game waiting its turn. This is not the case.

Each piece of music is generated dynamically using the same algorithm but because the logic is chaotic enough, the smallest difference in the initial values fed into it will result in a completely different tune being generated.

The routine responsible for creating this music is remarkably short so I've reproduced it here in full before we start to dive in and try to understand what's going on.

```
PlayTitleScreenMusic
        DEC baseNoteDuration
        BEQ MaybePlayNote
        RTS

MaybePlayNote
        LDA previousBaseNoteDuration
        STA baseNoteDuration

        DEC intervalBetweenNewNoteSets
```

```
        BNE MaybePlayVoice1

        LDA #$C0
        STA intervalBetweenNewNoteSets

        ; This is what will eventually time us out of playing
        ; the title music and enter attract mode.
        INC f7PressedOrTimedOutToAttractMode

        LDX baseIndexToMusicNoteArray
        LDA titleMusicNoteArray,X
        STA offsetForNextVoice1Note

        INX
        TXA
        AND #$03
        STA baseIndexToMusicNoteArray
        BNE MaybePlayVoice1

        JSR UpdateBaseNoteDurationAndTitleMusicNoteArray

MaybePlayVoice1
        DEC voice1NoteDuration
        BNE MaybePlayVoice2

        LDA #$30
        STA voice1NoteDuration

        LDX voice1IndexToMusicNoteArray
        LDA titleMusicNoteArray,X
        CLC
        ADC offsetForNextVoice1Note
        TAY
        STY offsetForNextVoice2Note

        JSR PlayNoteVoice1

        INX
        TXA
        AND #$03
        STA voice1IndexToMusicNoteArray

MaybePlayVoice2
        DEC voice2NoteDuration
        BNE MaybePlayVoice3

        LDA #$0C
        STA voice2NoteDuration
        LDX voice2IndexToMusicNoteArray
        LDA titleMusicNoteArray,X
        CLC
        ADC offsetForNextVoice2Note
        STA offsetForNextVoice3Note
        TAY
        JSR PlayNoteVoice2
        INX
        TXA
        AND #$03
        STA voice2IndexToMusicNoteArray

MaybePlayVoice3
        DEC voice3NoteDuration
        BNE ReturnFromTitleScreenMusic

        LDA #$03
        STA voice3NoteDuration

        LDX voice3IndexToMusicNoteArray
        LDA titleMusicNoteArray,X
        CLC
        ADC offsetForNextVoice3Note
        TAY
        JSR PlayNoteVoice3
        INX
        TXA
        AND #$03
        STA voice3IndexToMusicNoteArray
ReturnFromTitleScreenMusic
        RTS
```

Listing 1.1: Routine responsible for playing the title tune.

1.0.1 Some Basics

The rudiments of playing music on the Commodore 64 are simple. It has a powerful-for-its-time sound chip that has 3 tracks or 'voices'. You can play any note across 8 octaves on each of these voices together or separately. There are a whole bunch of settings you can apply to each voice to determine the way the note sounds. We'll cover a couple of these settings here but when it comes to playing music these extra settings aren't so important. They're much more useful when generating sound effects.

Playing a note on one of the voices consists of loading a two-byte value into the location (or 'register') associated with that voice. Here's the routine in Iridis used to play a note for the theme tune on Voice '1':

```
PlayNoteVoice1
        LDA #$21
        STA $D404     ;Voice 1: Control Register
        LDA titleMusicLowBytes,Y
        STA $D400     ;Voice 1: Frequency Control - Low-Byte
        LDA titleMusicHiBytes,Y
        STA $D401     ;Voice 1: Frequency Control - High-Byte
        RTS
```

Listing 1.2: Plays a note on Voice 1. The routine is supplied with a value in Y that indexes into two arrays containing the first (Hi) and second (Lo) byte respectively associated with the selected note.

Once the selected bytes have been loaded into `$D400` and `$D401` the new note will start playing. It's as blunt an instrument as that. (Well not quite, we'll cover some other gory details soon).

The full list of available notes is given in the C64 Progammer's Reference Manual. I've adapted and reproduced it below.

Octave	Note	High Byte	Low Byte	Octave	Note	High Byte	Low Byte	Octave	Note	High Byte	Low Byte
0	C	$01	$0C	2	G#	$06	$A7	5	E	$2A	$3E
0	C#	$01	$1C	2	A	$07	$0C	5	F	$2C	$C1
0	D	$01	$2D	2	A#	$07	$77	5	F#	$2F	$6B
0	D#	$01	$3E	2	B	$07	$E9	5	G	$32	$3C
0	E	$01	$51	3	C	$08	$61	5	G#	$35	$39
0	F	$01	$66	3	C#	$08	$E1	5	A	$38	$63
0	F#	$01	$7B	3	D	$09	$68	5	A#	$3B	$BE
0	G	$01	$91	3	D#	$09	$F7	5	B	$3F	$4B
0	G#	$01	$A9	3	E	$0A	$8F	6	C	$43	$0F
0	A	$01	$C3	3	F	$0B	$30	6	C#	$47	$0C
0	A#	$01	$DD	3	F#	$0B	$DA	6	D	$4B	$45
0	B	$01	$FA	3	G	$0C	$8F	6	D#	$4F	$BF
1	C	$02	$18	3	G#	$0D	$4E	6	E	$54	$7D
1	C#	$02	$38	3	A	$0E	$18	6	F	$59	$83
1	D	$02	$5A	3	A#	$0E	$EF	6	F#	$5E	$D6
1	D#	$02	$7D	3	B	$0F	$D2	6	G	$64	$79
1	E	$02	$A3	4	C	$10	$C3	6	G#	$6A	$73
1	F	$02	$CC	4	C#	$11	$C3	6	A	$70	$C7
1	F#	$02	$F6	4	D	$12	$D1	6	A#	$77	$7C
1	G	$03	$23	4	D#	$13	$EF	6	B	$7E	$97
1	G#	$03	$53	4	E	$15	$1F	7	C	$86	$1E
1	A	$03	$86	4	F	$16	$60	7	C#	$8E	$18
1	A#	$03	$BB	4	F#	$17	$B5	7	D	$96	$8B
1	B	$03	$F4	4	G	$19	$1E	7	D#	$9F	$7E
2	C	$04	$30	4	G#	$1A	$9C	7	E	$A8	$FA
2	C#	$04	$70	4	A	$1C	$31	7	F	$B3	$06
2	D	$04	$B4	4	A#	$1D	$DF	7	F#	$BD	$AC
2	D#	$04	$FB	4	B	$1F	$A5	7	G	$C8	$F3
2	E	$05	$47	5	C	$21	$87	7	G#	$D4	$E6
2	F	$05	$98	5	C#	$23	$86	7	A	$E1	$8F
2	F#	$05	$ED	5	D	$25	$A2	7	A#	$EE	$F8
2	G	$06	$47	5	D#	$27	$DF	7	B	$FD	$2E

Figure 1.1: All available notes on the C64 and their corresponding hi/lo byte values. Note that Iridis Alpha only uses octaves 3 to 7. The available notes in octaves 1 to 2 are never used.

With 96 notes in total available, Iridis only uses 72 of them, omitting the 2 lowest octaves. We can see this when we look at the note table in the game. This pair of arrays are where the title music logic plucks the note to be played once it has dynamically selected one:

```
                        ;       C   C#  D   D#  E   F   F#  G   G#  A   A#  B
titleMusicHiBytes       .BYTE $08,$08,$09,$09,$0A,$0B,$0B,$0C,$0D,$0E,$0E,$0F  ; 4
                        .BYTE $10,$11,$12,$13,$15,$16,$17,$19,$1A,$1C,$1D,$1F  ; 5
                        .BYTE $21,$23,$25,$27,$2A,$2C,$2F,$32,$35,$38,$3B,$3F  ; 6
                        .BYTE $43,$47,$4B,$4F,$54,$59,$5E,$64,$6A,$70,$77,$7E  ; 7
                        .BYTE $86,$8E,$96,$9F,$A8,$B3,$BD,$C8,$D4,$E1,$EE,$FD  ; 8

                        ;       C   C#  D   D#  E   F   F#  G   G#  A   A#  B
titleMusicLowBytes      .BYTE $61,$E1,$68,$F7,$8F,$30,$DA,$8F,$4E,$18,$EF,$D2  ; 4
                        .BYTE $C3,$C3,$D1,$EF,$1F,$60,$B5,$1E,$9C,$31,$DF,$A5  ; 5
                        .BYTE $87,$86,$A2,$DF,$3E,$C1,$6B,$3C,$39,$63,$BE,$4B  ; 6
                        .BYTE $0F,$0C,$45,$BF,$7D,$83,$D6,$79,$73,$C7,$7C,$97  ; 7
                        .BYTE $1E,$18,$8B,$7E,$FA,$06,$AC,$F3,$E6,$8F,$F8,$2E  ; 8
```

Listing 1.3: The lookup table for all of the notes used in the theme music. The two lowest available octaves are not used by the game. To see this for yourself compare the first entry in `titleMusicHiBytes`/`titleMusicLowBytes` ($08 and $61 giving $0861) with the entry highlighted in red in the previous table.

So now that we know where the notes are and how to make them go beep we just have to figure out the order that `PlayTitleScreenMusic` contrives to play them.

It would certainly help if we could see what the music looks like, so lets do that. Here is the opening title tune as sheet music in Western notation.

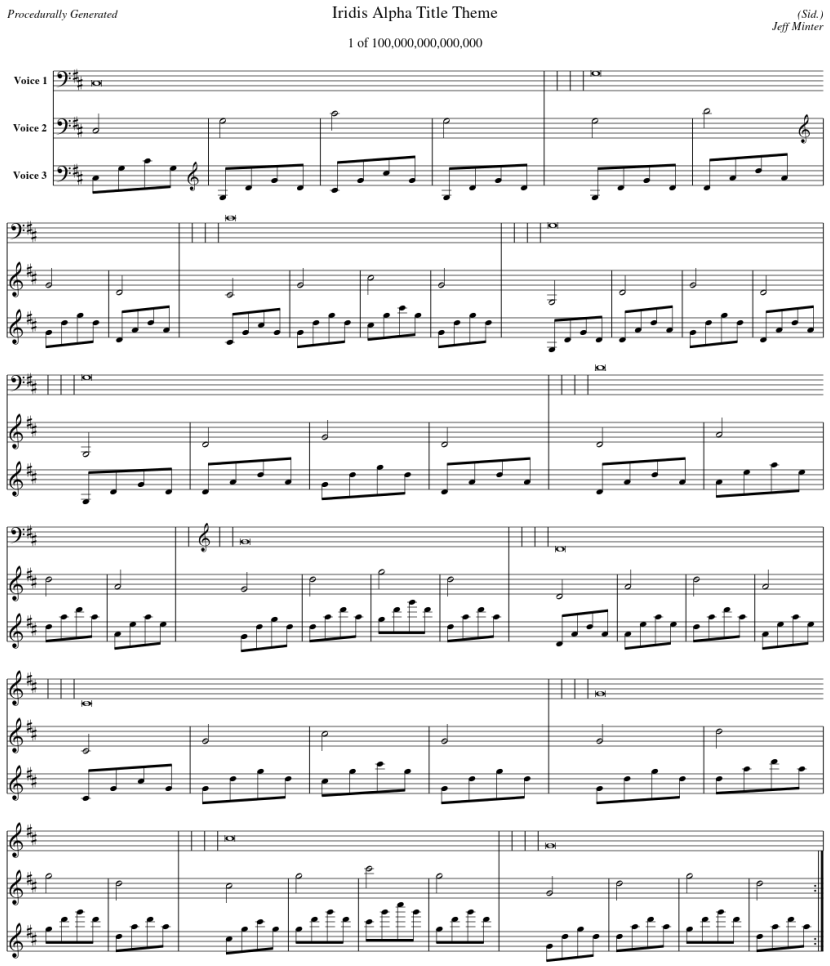
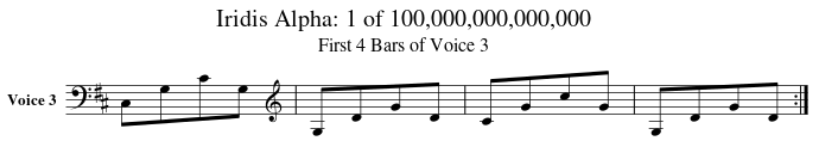


Figure 1.2: The first title tune in Iridis Alpha.

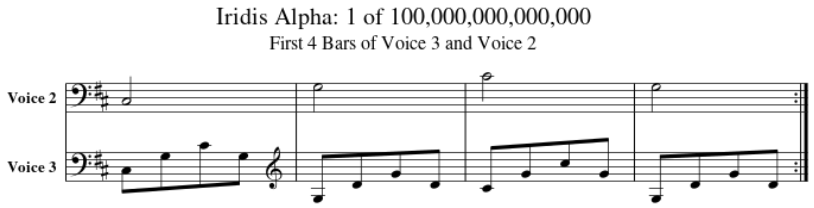
Structure

Even if you can't read sheet music notation some structure should be evident.

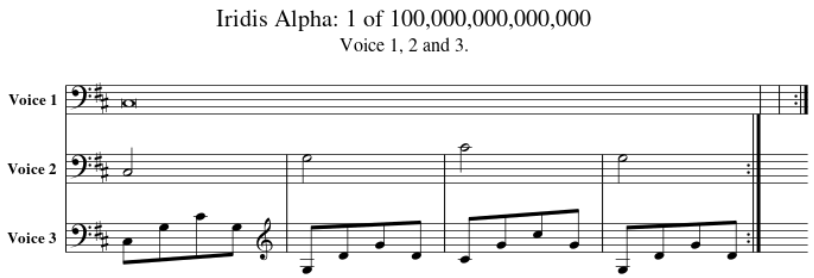
Voice 3 carries the main melody.



For every 4 notes Voice 3 plays, Voice 2 chimes in with a new note that it sustains until the next one.



Voice 1 does the same for every 16 notes that Voice 3 plays and every 4 notes of Voice 2..



Armed with this insight we can see it reflected in the logic in `PlayTitleScreenMusic`. This routine is called regularly by a system interrupt, a periodic wake-up call performed

by the C64 CPU. So multiple times every second it is run and must figure out what new notes, if any, to play on each of the three voices.

Here it is deciding whether or not to play new note on Voice 1:

```
MaybePlayVoice1
        DEC voice1NoteDuration
        BNE MaybePlayVoice2

        LDA #$30
        STA voice1NoteDuration
```

Listing 1.4: `MaybePlayVoice1` part of `PlayTitleScreenMusic`.

`voice1NoteDuration` is used to count the interval between notes on Voice 1. It's decremented on each visit and when it reaches zero it gets reset to 48 ($30) and a note is played. What's being counted here isn't seconds, it's cycles or 'interrupts'. So this translates to only a few seconds between notes being played.

The same is done for both Voice 2 and Voice 3 but the intervals are shorter: 12 ($0C) and 3 ($03). This matches the relationship we see in the sheet music, one note in Voice 1 for every sixteen in Voice 3 (48/3=16) and one note in Voice 2 for every four in Voice 3 (12/3=4).

```
MaybePlayVoice2
        DEC voice2NoteDuration
        BNE MaybePlayVoice3

        LDA #$0C
        STA voice2NoteDuration
```

Listing 1.5: `MaybePlayVoice2` part of `PlayTitleScreenMusic`.

```
MaybePlayVoice3
        DEC voice3NoteDuration
        BNE ReturnFromTitleScreenMusic

        LDA #$03
        STA voice3NoteDuration
```

Listing 1.6: `MaybePlayVoice3` part of `PlayTitleScreenMusic`.

Phrasing

Now that we've identified the underlying 4-bar structure of the arrangement. We can take a closer look at the phrasing of the invidividual parts. Voice 3 has a simple repetitive structure for each 4-bar phrase:

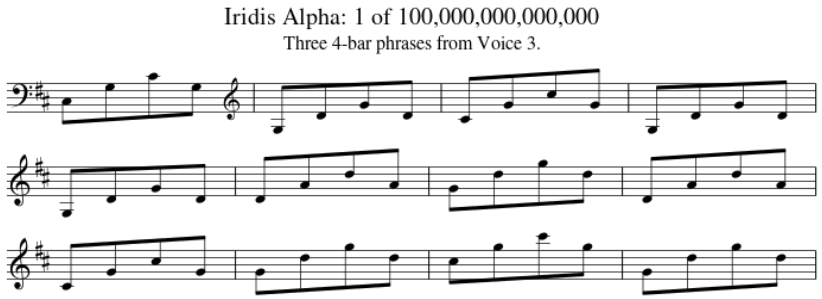
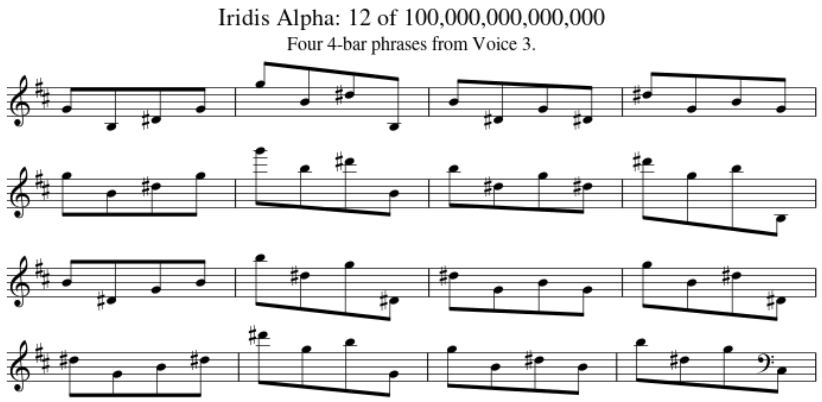


Figure 1.3: Bars 2 and 4 are always repeated

Bars 2 and 4 are repeated. Each bar consists of the same tonic formula: three notes rising two notes at a time, falling back on the final note. The difference between bars 1 and 3 is a simple key change.

As new tunes are generated the simple repetion seen in the first iteration disappears. Here the first 16 bars for Voice 3 from iterations 12 and 18:



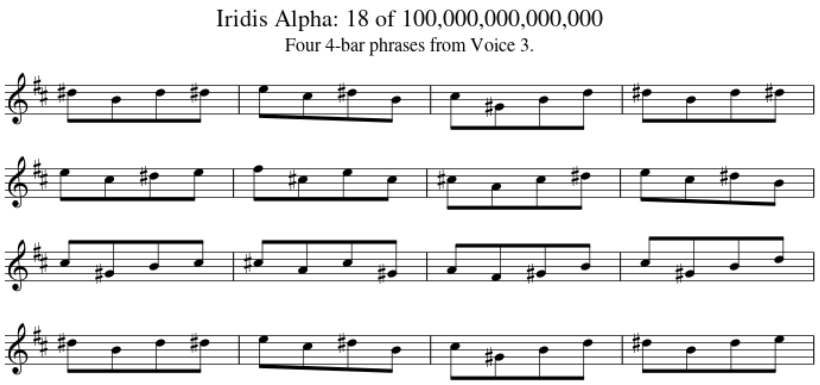


Figure 1.4: Key changes and same basic structure, but no repetitions.

Diversion: Extracting the Title Music

Since each tune is dynamically generated there's nowhere for us to pull them from. We could record the tunes as audio files and maybe extract something useful that way. A feature of Vice, the C64 emulator, allows us to do something much simpler. We can log every note that's played to a text file and use that trace to reconstruct the tunes.

We launch Iridis Alpha with `x64` using the following command:

```
x64 -moncommands moncommands.txt orig/iridisalpha.prg
```

The `moncommands.txt` file contains a series of debugger directives that tells `x64` to log every value stored to the music registers at `$D400-$D415`. This will capture all notes played on all three voices as well as any updates made to the other sound parameters and write them to `IridisAlphaTitleMusicAll.txt`:

```
log on
logname "IridisAlphaTitleMusicAll.txt"
tr store D400 D415
```

We end up with `IridisAlphaTitleMusicAll.txt` full of lines like:

```
TRACE: 1  C:$d400-$d415  (Trace store)
#1 (Trace store d400)  279 052
.C:1598  8D 00 D4    STA $D400      - A:61 X:00 Y:00 SP:e8 ..-..I..   96135469
#1 (Trace store d401)  279 060
.C:159e  8D 01 D4    STA $D401      - A:08 X:00 Y:00 SP:e8 ..-..I..   96135477
#1 (Trace store d40b)  280 059
```

This examples gives us the value in `A` written to each register for Voice 1. For example, `$61` has been written to `$D400` and `$08` has been written to `$D401`.

We can now write a short Python notebook that parses this file and for each tune constructs three arrays, each representing a voice, with the sequence of notes played to each. For example, in the extract above we can extract `$0861` as the note 'C' in octave 3 played on Voice 1 (`$D400`–`$D401`). (Refer to the tables above to see why `$0861` translates to 'C-3'.

With the sequence of notes in three arrays, each representing one of the 3 voices, it is a simple matter to transform this into ABC format, a music notation frequently used for traditional music.

```
%abc-2.2
%%pagewidth 30cm
%%header "Example  page: $P"
%%footer " $T"
%%gutter .5cm
%%barsperstaff 16
%%titleformat R-P-Q-T C1 O1, T+T N1
%%composerspace 0
X: 2 % start of header
T: Iridis Alpha Title Theme
T:1 of 100,000,000,000,000
C: (Sid.)
O: Jeff Minter
R: Procedurally Generated
L: 1/8
K: D % scale: C major
V:1 name="Voice 1"
C,16     |      |      |      | G,16     |      |      |      | C16      |      |      |      | G,16     |      |      |      |
       G,16     |      |      |      | D16      |      |      |      | G16      |      |      |      | D16      |      |      |      |
         | C16      |      |      |      | G16      |      |      |      | c16      |      |      |      | G16      |      |      |      |
           :|
V:2 name="Voice 2"
C,4      | G,4      | C4       | G,4      | G,4      | D4       | G4       | D4       | C4       | G4       | c4       | G4       | G,4      | D4
           | G4       | D4       | G,4      | D4       | G4       | D4       | D4       | A4       | d4       | A4       | G4       | d4       | g4
           | d4       | D4       | A4       | d4       | A4       | C4       | G4       | c4       | G4       | G4       | d4       | g4       | d4
         | c4       | g4       | c'4      | g4       | G4       | d4       | g4       | d4       :|
V:3 name="Voice 3"
C,1G,1C1G,1|G,1D1G1D1|C1G1c1G1|G,1D1G1D1|G,1D1G1D1|D1A1d1A1|G1d1g1d1|D1A1d1A1|C1G1c1G1|G1d1g1d1|c1g1c'1g1|
       G1d1g1d1|G,1D1G1D1|D1A1d1A1|G1d1g1d1|D1A1d1A1|G,1D1G1D1|D1A1d1A1|G1d1g1d1|D1A1d1A1|D1A1d1A1|A1e1a1e1|
       d1a1d'1a1|A1e1a1e1|G1d1g1d1|d1a1d'1a1|g1d'1g'1d'1|d1a1d'1a1|D1A1d1A1|A1e1a1e1|d1a1d'1a1|A1e1a1e1|C1G1c1G1
       |G1d1g1d1|c1g1c'1g1|G1d1g1d1|G1d1g1d1|d1a1d'1a1|g1d'1g'1d'1|d1a1d'1a1|c1g1c'1g1|g1d'1g'1d'1|c'1g'1c''1g'
       1|g1d'1g'1d'1|G1d1g1d1|d1a1d'1a1|g1d'1g'1d'1|d1a1d'1a1:|
```

Listing 1.7: Title Tune No 1 in ABC format

We can the use the tool 'abcm2ps' to transform this into an SVG image file giving the music in standard Western notation.